

# Object-oriented C++

CODERS SCHOOL

<https://coders.school>



Łukasz Ziobroń  
lukasz@coders.school

# Łukasz Ziobroń

Not only programming experience:

- C++ and Python developer @ Nokia & Credit Suisse
- Scrum Master @ Nokia & Credit Suisse
- Code Reviewer @ Nokia
- Webmaster (HTML, PHP, CSS) @ StarCraft Area

Training experience:

- C++ trainings @ Coders School
- Practical Aspects Of Software Engineering @ PWr, UWr
- Nokia Academy @ Nokia
- Internal corporate trainings

Public speaking experience:

- Academic Championships in Team Programming
- code::dive conference
- code::dive community



# Agenda

- 4 pillars of objectivity
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

# 4 pillars of objectivity

- Abstraction
- Enkapsulation (hermetization)
- Inheritance
- Polymorphism



*git clone <https://github.com/ziobron/Cars.git>*

# Abstraction

- Interface
  - The public part of a class
    - Member functions - obvious
    - Non-members functions
    - Member types
    - Member fields
    - Template parameters
    - Specializations
  - Example: [std::vector on cppreference.com](http://en.cppreference.com/std/vector)
  - The private part (implementation) is unknown
- Object Oriented Design (OOD)

*Make interfaces easy to use correctly and hard to use incorrectly*

~ Scott Meyers, [Effective C++](#)

# Bad interface example

// A date class which is easy to use but also easy to use wrong.

```
class Date
```

```
{
```

```
public:
```

```
    Date(int month, int day, int year);
```

```
    ...
```

```
};
```

// Both are ok, but some european programmer may use it wrong.

// Because european time is dd/mm/yyyy instead of mm/dd/yyyy.

```
Date d(3, 4, 2000);
```

```
Date d(4, 3, 2000);
```

# Encapsulation

- Access specifiers
  - public – struct default
  - protected
  - private – class default
- Setters and getters
- Unnamed namespace

# Inheritance

- Constructors and destructors call order
  - Constructors – base class first, then derived
  - <https://ideone.com/Kgb46n>
- Diamond problem
  - virtual inheritance
- Class from struct inheritance is...
  - private - <https://ideone.com/Rdd6Uf>
- Struct from class inheritance is...
  - public - <https://ideone.com/x460vN>



# Inheritance access modifiers

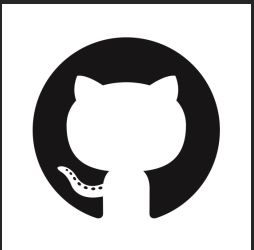
	public	protected	private
public	public	protected	private
protected	protected	protected	private
private	private	private	private

# Polymorphism

- Virtual functions
- Pure virtual functions (=0)
- Abstract classes
  - At least one pure virtual function
- vtable and vptr
  - implementation of polymorphism
  - constructor of derived class overrides base class records in vtable

# Exercise: Cars

1. Design proper abstraction (interfaces)
2. Apply inheritance
3. Fix encapsulation
4. Use polymorphism to represent every type of car, using a single pointer
5. Fix a diamond problem
6. Fix potential memory leaks
7. Think about the way of keeping engines in cars. Should they be kept by a value, reference or a pointer (what kind of pointer)?
8. Is this code testable?



*git clone <https://github.com/coders-school/Cars.git>*

# Post-work

You can work in groups or individually. Please fork the repo and submit a Pull Request after you have finished.

1. Create `InvalidGear` exception. It should be thrown when someone tries eg. change a gear from 5 to R. It should inherit from one of STL exceptions
2. Fix interfaces to be easy to use correctly and hard to use incorrectly (like `accelerate(-999)`)
3. (Optional) Write proper unit tests to this code.
4. Read one of below articles. It will be useful for the next lesson.
  - [SOLID czyli dobre praktyki w programowaniu obiektowym](#) (in Polish)
  - [S.O.L.I.D: The First 5 Principles of Object Oriented Design](#) (in English)

# CODERS SCHOOL

<https://coders.school>



Łukasz Ziobroń  
[lukasz@coders.school](mailto:lukasz@coders.school)